A Kleene theorem for Petri automata

Paul Brunet

A Kleene theorem for Petri automata

Paul Brunet
Plume team – LIP, Université de Lyon, CNRS, ENS de Lyon, Inria, UCBL
paul.brunet@ens-lyon.fr

Abstract

While studying the equational theory of Kleene Allegories (KAl), we recently proposed two ways of defining sets of graphs [BP15]: from KAl expressions, that is, regular expressions with intersection and converse; and from a new automata model, Petri automata, based on safe Petri nets. To be able to compare the sets of graphs generated by KAl expressions, we explained how to construct Petri automata out of arbitrary KAl expressions.

In the present paper, we describe a reverse transformation: recovering an expression from an automaton. This has several consequences. First, it generalises Kleene theorem: the graph languages specified by Petri automata are precisely the languages denoted by KAl expressions. Second, this entails that decidability of the equational theory of Kleene Allegories is equivalent to that of language equivalence for Petri automata. Third, this transformation may be used to reason syntactically about the occurrence nets of a safe Petri net, provided they are parallel series.

Keywords regular expressions, graph language, Petri nets, finite automata, allegories, intersection, converse, Kleene theorem

1. Introduction

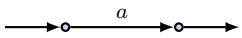
Petri Automata are Petri net-based automata, and are equipped with an operational semantics allowing them to recognise sets of graphs. They were introduced in a recent paper [2] to study Kleene Allegory expressions: terms built from a finite alphabet of variables, with the constants 0 and 1, the unary operators converse and Kleene star, and the binary operators union, composition and intersection. To any such expression, one can associate a set of Two Terminal Series Parallel graphs (called graphs in the sequel) such that two expressions are universally equivalent when interpreted as binary relations if and only if their associated sets of graphs are equal. In [2] we gave a method to build from any Kleene Allegory expression a Petri automaton recognising the appropriate set of series parallel graphs.

It is then natural to wonder what is the class of graph languages recognised by Petri automata. For the usual notion of finite state automata, the well known Kleene Theorem states that the languages recognisable by automata are exactly those specifiable by regular expressions. In this paper we provide a similar theorem for Petri automata and Kleene Allegory expressions.

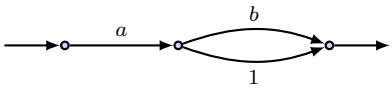
In the remainder of this introduction, we explain informally what are the graphs associated with expressions and Petri automata and we give the outline of the computation of the expression corresponding to a given automaton.

1.1 Expressions

For the bulk of the development, we restrict ourselves to a simpler setting: we consider expressions over the syntax of Kleene lattices, that is to say without the converse operation. To any such expression e, we associate a set $\mathscr{G}(e)$ of labelled directed graphs with two distinguished vertices understood as input and output. A letter a is associated to a singleton set containing the following graph:



The input in this graph is the vertex with the unmarked incoming arrow, and the output is the other vertex, with the outgoing arrow. Similarly the constant 1 yields a singleton set containing the graph where the input is linked to the output by an edge labelled 1. We define the sequential composition of two such graphs by identifying the output of the first graph with the input of the second one. We also build the parallel composition (corresponding to the intersection operation) of two graphs by identifying both their inputs and their outputs. For instance $\mathscr{G}(a \cdot (b \cap 1))$ simply contains the graph:



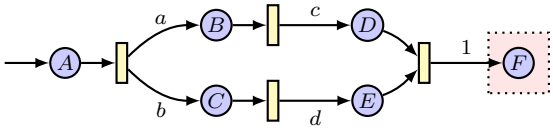
With these constructs we can define the set of graphs associated to an expression by structural induction: $\mathscr{G}(1)$ and $\mathscr{G}(a)$ are defined as before, $\mathscr{G}(e \cup f)$ is the union of $\mathscr{G}(e)$ and $\mathscr{G}(f)$; $\mathscr{G}(0)$ is the empty set; $\mathscr{G}(e \cdot f)$ (respectively $\mathscr{G}(e \cap f)$) is the set of graphs obtained by composing in sequence (resp. in parallel) a graph in $\mathscr{G}(e)$ with a graph in $\mathscr{G}(f)$; and finally the graphs in $\mathscr{G}(e^\star)$ are either the graph of 1 or built by sequentially composing any finite number of graphs from $\mathscr{G}(e)$.

1.2 Petri automata

A Petri automaton over the alphabet Σ is defined by:

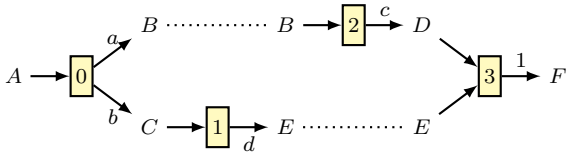
- a finite set of places (denoted in graphical representations by round vertices),
- a finite set $\mathscr{T}$ of transitions (denoted by rectangular vertices). Each transition is equipped with a set of input places (incoming arrows) and a set of outputs. These outputs are pairs of a label in $\Sigma \cup \{1\}$, and a place.
- an initial place and a finite set of final configurations, a configuration being a set of places.

Here is an example of Petri automaton:

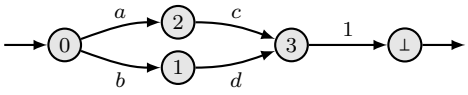


The initial place is A, as denoted by the unmarked incoming arrow. There is a single final configuration, containing only the place F (as indicated by the red dotted rectangle). In this paper, we will restrict ourselves to automata having a single final configuration, composed of one place.

Runs in such an automaton are plays of a token firing game: one starts by placing tokens on certain places, and if every place in the input of a transition holds a token, this transition is *enabled* and may be fired. Firing a transition will then remove one token from each of its inputs, and place one on each output. If the run starts with a single token on the initial place, we call it *initial*, and if it stops with one token on each place of a final configuration it is *final*. An *accepting* run is both initial and final. For instance, here is a graphical representation of an accepting run in the previous automaton:



We further impose that Petri automata should be *safe*, meaning that at any step of any initial run of the automaton at most one token is present on each place. The last step to get the language of an automaton is to generate a graph from any run of the automaton (called the *trace* of the run). We do so by considering a graph whose vertices are the transitions (plus an additional final vertex), and labelled edges are extrapolated from the outputs of the transitions. For instance the run from the previous example yields the graph:



We may thus define the language $\mathscr{G}(\mathscr{A})$ of an automaton $\mathscr{A}$ as the set of the traces of its accepting runs. In this paper we only consider automata whose language contain exclusively parallel series graphs (because we are only interested in such graphs). As a byproduct of the construction presented here we show that this property is decidable.

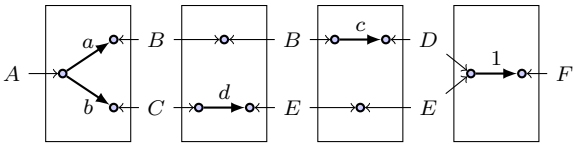
Remark 1. The definitions we give here are sometimes different from those stated in [2], but the results apply nonetheless. We discuss these differences in Section 5.1.

1.3 Main result

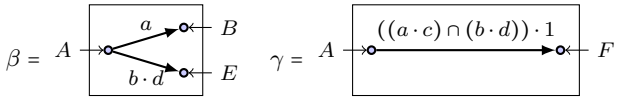
The main technical result of this paper is a construction allowing one to build from an automaton $\mathscr{A}$ an expression $\mathscr{E}(\mathscr{A})$ such that:

$$\mathscr{G}(\mathscr{A}) = \mathscr{G}(\mathscr{E}(\mathscr{A})).$$

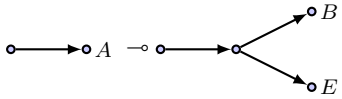
Before properly describing the construction, we need a compositional way to represent a local and partial view of a graph. This will be achieved through the notions of boxes and types. In the same way a sequence of letters may be a factor of some word, boxes are meant to represent "slices" of graphs. Formally, if A and B are sets of places, a box going from A to B is a labelled directed graph G with maps from A to the vertices of G and from B to the vertices of G. For instance, in the run presented before as an example, we may associate a box to each transition. This yields the following sequence of boxes:



We may compose these boxes by merging the outputs of the first box with the inputs of the second box, and reducing the graph by applying the operators $\cdot$ and $\cap$ on edges. For instance the first two boxes above compose to produce the box β, and the whole sequence can be composed as the box γ:

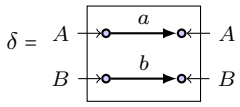


Types in this setting are trees with leaves labelled with places. We can type a box β with a pair of trees $\sigma \multimap \tau$ if the tree σ composed with the box β yields the tree τ. For instance the box β has the type:

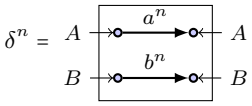


We build a finite state automata whose states are the types, and whose transitions between the states σ and τ are labelled with the boxes built from transitions of the Petri automaton with type $\sigma \multimap \tau$. Using the standard Kleene Theorem, this allows us to produce a regular expression over boxes $e_{\mathscr{A}}$. By changing the set of letters to finite sets of boxes labelled with expressions, we are then able to internalise all of the regular operations. This produces in the end a finite number of boxes with one input (corresponding to the initial place) and one output (corresponding to the final place), linked by an edge labelled with a expression. The union of these expressions is the expression $\mathscr{E}(\mathscr{A})$, that denotes the same set of graphs as $\mathscr{A}$.

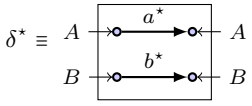
The technical difficulty here is the computation of the star of a set of boxes. For instance, consider the box:



Its star yields all boxes of the following shape, for $n \in \mathbb{N}$:



This set of boxes cannot be represented by a finite number of boxes. However, one must realise that because the two branches of the box δ are disconnected, they must correspond to concurrent process in the Petri automaton, and thus can be iterated separately. Intuitively, the net cannot force two independent processes to perform exactly the same number of iterations. Because of this, the following box is the right choice to represent the star of the box δ:



1.4 Outline

We begin the paper by introducing in Section 2 the expressions and automata we consider here, and the sets of graphs they represent. After a few definitions and results about graphs, we detail in Section 3 the types and boxes we use, and we establish some facts

about them. Section 4 is dedicated to the main contribution of the paper, the construction extracting expressions from Petri automata. Finally we list some consequences of this result, as well as some links with previous work, in Section 5.

2. Petri automata & expressions

Regular expressions and rationnal languages We recall a few standard definitions and notations about regular expressions. A *regular expression* over the alphabet Σ is a term built from variables taken in Σ, the constants 1 and 0, the binary operators $\cdot$ and $\cup$, and the unary $\_^\star$. The set is denoted by $\mathrm{Reg}\langle\Sigma\rangle\Sigma$. The *language denoted by* e, written $[\![e]\!]$, is a set of words over Σ defined inductively by:

$$[\![a]\!] \coloneqq \{a\}\,; \quad [\![1]\!] \coloneqq \{\epsilon\}\,; \quad [\![0]\!] \coloneqq \varnothing; \quad [\![e \cup f]\!] \coloneqq [\![e]\!] \cup [\![f]\!];$$
$$[\![e \cdot f]\!] \coloneqq \{u \cdot v \mid u \in [\![e]\!] \text{ and } v \in [\![f]\!]\}\,;$$
$$[\![e^\star]\!] \coloneqq \{u_1 \cdot \ldots \cdot u_n \mid n \in \mathbb{N},\ \forall i, u_i \in [\![e]\!]\}\,.$$

(ϵ being the empty word.)

If $[\![e]\!] = [\![f]\!]$, we say that regular laws prove the equality of e and f, written $KA \vDash e = f$.

2.1 Expressions

Expressions $e, f \ldots$ are built on the signature $\langle 1, \cdot, \cup, \cap, \_^+\rangle$[1]. We denote by $\mathrm{Reg}^{\cap}_{\Sigma}$ the set of expressions over the finite alphabet Σ. A 2-pointed labelled directed graph G over $\Sigma_\bullet \coloneqq \Sigma \cup \{1\}$, simply called graph in the sequel, is given by a tuple $\langle V, E, \iota, o\rangle$ with a finite set of vertices V, a set of labelled edges $E \subseteq V \times \Sigma_\bullet \times V$, and $\iota, o \in V$ two distinguished vertices, respectively called input and output. The sequential composition (written $\_\cdot\_$) of two graphs with disjoint sets of vertices can be performed by identifying the output of the first graph and the input of the second one. Their parallel composition (written $\_ \cap \_$) consists in merging their inputs and merging their outputs. See Figure 1 for a graphical description of these constructions.

We assign to each expression a set of such graphs, called the graph language of the expression.

Definition 2 (Graph language of an expression: $\mathscr{G}(e)$)
The graph language is defined by structural induction as follows:

$$\mathscr{G}(a) \coloneqq \left\{ \xrightarrow{\ a\ } \right\}$$
$$\mathscr{G}(1) \coloneqq \left\{ \xrightarrow{\ 1\ } \right\}$$
$$\mathscr{G}(0) \coloneqq \varnothing$$
$$\mathscr{G}(e \cup f) \coloneqq \mathscr{G}(e) \cup \mathscr{G}(f)$$
$$\mathscr{G}(e \cdot f) \coloneqq \{E \cdot F \mid E \in \mathscr{G}(e) \text{ and } F \in \mathscr{G}(f)\}$$
$$\mathscr{G}(e \cap f) \coloneqq \{E \cap F \mid E \in \mathscr{G}(e) \text{ and } F \in \mathscr{G}(f)\}$$
$$\mathscr{G}(e^+) \coloneqq \{E_1 \cdot \ldots \cdot E_n \mid n > 0,\ \forall i, E_i \in \mathscr{G}(e)\}$$

*

Those graphs were introduced independently by Freyd and Scedrov [3, page 208], and Andréka and Bredikhin [1] (except for the way 1 is handled). Notice that the graphs produced by this construction are exactly Two Terminal Series Parallel graphs labelled with $\Sigma_\bullet$ [8].

2.2 Petri automata

A Petri automaton is a Petri net whose transition's outputs are labelled by the set $\Sigma_\bullet$.

[1] Notice that we don't use the Kleene star, but rather its positive variant. It is more convenient here to have $e^\star$ be defined as $1 \cup e^+$.

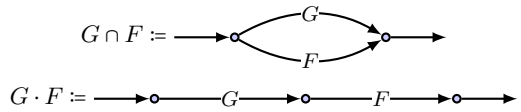


Figure 1: Elementary graph constructions

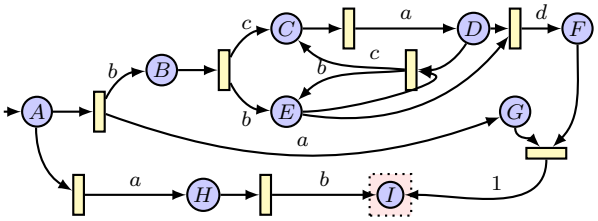


Figure 2: A Petri automaton. The initial place is A, and the final configurations is $\{I\}$.

Definition 3 (Petri Automaton)
A *Petri automaton* $\mathscr{A}$ over the alphabet Σ is a tuple $\langle P, \mathscr{T}, \iota, f\rangle$ where:

- P is a finite set of *places*,
- $\mathscr{T} \subseteq \mathcal{P}(P) \times \mathcal{P}(\Sigma_\bullet \times P)$ is a set of *transitions*,
- $\iota \in P$ is the *initial place* of the automaton,
- $f \in P$ is the *final place* of the automaton.

For each transition $t = (\underline{t}, \overline{t}) \in \mathscr{T}$, $\underline{t}$ and $\overline{t}$ are assumed to be non-empty; $\underline{t} \subseteq P$ is the *input of* t; and $\overline{t} \subseteq \Sigma_\bullet \times P$ is the *output of* t. *

This definition of Petri automata correspond to what we call *simple Petri automata* in [2]. However, this case is general enough, as discussed in Section 5.1. We use the graphical notation from the introduction to represent Petri automata; the Petri automaton from Figure 2 will be used as a running example.

From a configuration $\xi \subseteq P$, a transition $t = (\underline{t}, \overline{t}) \in \mathscr{T}$ is *enabled* if $\underline{t} \subseteq \xi$. If so, one may *fire* t, which produces a new configuration ξ' defined by

$$\xi' = \xi \setminus \underline{t} \cup \left\{p \in P \;\middle|\; \exists x \in \Sigma : (x, p) \in \overline{t}\right\}.$$

We write $\xi \xrightarrow{t}_{\mathscr{A}} \xi'$ in this case.

In the sequel, we assume all considered Petri automata to be *safe*. (I.e., in Petri nets terminology, such that any reachable marking has at most one token in each place [6]). Formally, with our definitions: a Petri automaton $\langle P, \mathscr{T}, \iota, f\rangle$ is safe if for all configuration $\xi \subseteq P$ reachable from $\{\iota\}$ by firing any number of transitions, if $(\underline{t}, \overline{t}) \in \mathscr{T}$ is enabled from ξ, $p \in \xi$, and $(x, p) \in \overline{t}$, then $p \in \underline{t}$.

Now we recall how to use Petri automata to define languages of graphs. We first define the *runs* of an automaton.

Definition 4 (Run, accepting run)
A *run* ξ in $\mathscr{A} = \langle P, \mathscr{T}, \iota, F\rangle$ is a sequence $\langle (\xi_k)_{0 \leqslant k \leqslant n}, (t_k)_{0 \leqslant k < n}\rangle$ of configurations and transitions, such that $\xi_k \subseteq P$, $t_k \in \mathscr{T}$ and $\forall k < n$, $\xi_k \xrightarrow{t_k}_{\mathscr{A}} \xi_{k+1}$. When $\xi_0 = \{\iota\}$ and $\xi_n = \{f\}$, we call ξ an *accepting run*. *

(Note that a run ξ is uniquely determined by ξ_0 and the sequence (t_k): all subsequent configurations can be computed deterministically.)

Example 5
Consider the following sequence of transitions from the automaton

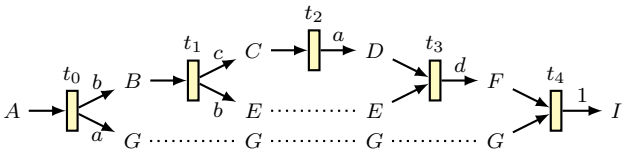


Figure 3: An accepting run in the automaton from Figure 2.

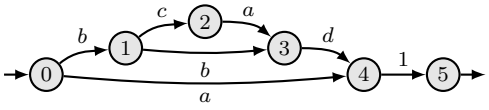


Figure 4: Trace of the run depicted in Figure 3.

in Figure 2:

$$\xi = \langle (\xi_0, \xi_1, \xi_2, \xi_3, \xi_4, \xi_5), (t_0, t_1, t_2, t_3, t_4) \rangle,$$

with

$$\begin{array}{l|l} \xi_0 = \{A\}, & t_0 = (\{A\}, \{(b,B),(a,G)\}), \\ \xi_1 = \{B,G\}, & t_1 = (\{B\}, \{(c,C),(b,E)\}), \\ \xi_2 = \{C,E,G\}, & t_2 = (\{C\}, \{(a,D)\}), \\ \xi_3 = \{D,E,G\}, & t_3 = (\{D,E\}, \{(d,F)\}), \\ \xi_4 = \{F,G\}, & t_4 = (\{F,G\}, \{(1,I)\}). \\ \xi_5 = \{I\}. & \end{array}$$

We have:

$$\{A\} \xrightarrow{t_0} \{B,G\} \xrightarrow{t_1} \{C,E,G\} \xrightarrow{t_2} \{D,E,G\} \xrightarrow{t_3} \{F,G\} \xrightarrow{t_4} \{I\},$$

and since $\{A\}$ is the initial configuration and $\{I\}$ is the final one, this sequence is an accepting run. It can be represented graphically as in Figure 3. ■

In [2], the language of a Petri automaton is defined via an operational semantics, describing the way an automaton "reads" a graph. For the sake of clarity, we use here a different definition, more directly related to the set of accepting runs of the automaton. We associate to each accepting run ξ a 2-graph, written $\mathscr{G}(\xi)$, called the *trace* of ξ. The language of the automaton is then defined as the set of traces of its accepting runs.

The trace is constructed by creating a vertex k for each transition $t_k = \left(\underline{t_k}, \overline{t_k}\right)$ of the run, plus a final vertex n. We add an edge (k, x, l) whenever there is some place q such that $(x, q) \in \overline{t_k}$, and t_l is the first transition after t_k in the run with q among its inputs, or $l = n$ if there is no such transition in the run.

Definition 6 (Trace of a run)
Let $\xi = \langle (\xi_k)_{0 \leqslant k \leqslant n}, \left(\underline{t_k}, \overline{t_k}\right)_{0 \leqslant k < n} \rangle$ be run. For an index $k \leqslant n$ and a place q, let $\nu(k,q)$ be either the smallest index l such that $k \leqslant l$ and $q \in \underline{t_l}$, or n if there is no such index.

The *trace of* ξ is the graph $\mathscr{G}(\xi) \coloneqq \langle \{0, \ldots, n\}, E_\xi, 0, n \rangle$ with

$$E_\xi \coloneqq \left\{ (k, x, \nu(k+1, q)) \;\middle|\; (x,q) \in \overline{t_k} \right\}.$$

*

We write $\mathscr{G}(\mathscr{A})$ for the language of a Petri automaton $\mathscr{A}$, defined as the set of traces of its accepting runs. The trace of the run presented in Figure 3 is depicted in Figure 4.

Remark 7. $\mathscr{G}(\mathscr{A})$ is different from $\mathscr{L}(\mathscr{A})$, as it is defined in [2]. See section Section 5.1 for a more detailed discussion.

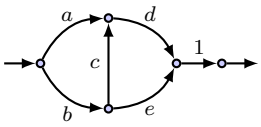


Figure 5: Non series parallel graph.

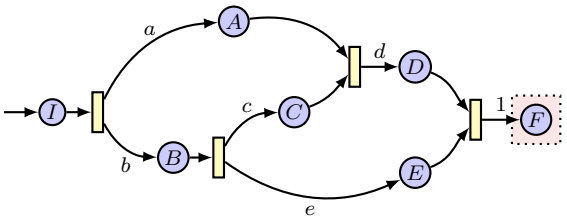


Figure 6: Non-SP Petri automaton.

Theorem 8 ([2, Theorem 17])**.** *For any expression $e \in Reg^{\cap}_{\Sigma}$, there is a Petri automaton $\mathscr{A}(e)$ such that*

$$\mathscr{G}(e) = \mathscr{G}(\mathscr{A}(e)).$$

2.3 SP-Petri automata

When trying to build expressions out of Petri automata, one immediately encounters a problem: traces of accepting runs are not necessarily series parallel graphs, which means they cannot be generated by any expression from $\mathrm{Reg}^{\cap}_{\Sigma}$. For instance the automaton given in Figure 6 has a single accepting run, whose trace is the graph draw in Figure 5. To circumvent this problem, we consider only well-behaved automata: automata whose accepting runs have parallel-series graphs.

Definition 9 (SP-automata)
An automaton $\mathscr{A}$ is said to be an *SP-automaton* if for any accepting run ξ in $\mathscr{A}$, $\mathscr{G}(\xi)$ is an SP-graph. *

Despite its infinitary formulation, this property is decidable. In fact, the procedure we describe later on would fail (in finite time) if it is given a non-SP automaton: hence it may be used to decide this property.[2]

3. Boxes

Notations We call *label space* a structure $\langle \mathbb{L}, \cdot, \cap \rangle$ such that $\cdot$ and $\cap$ are two internal associative binary operations, and $\cap$ is commutative. Examples of such spaces include:

$\mathbb{W}_X$ **:** the set of SP-graphs labelled by X. It is the free label space over a finite set of atomic variables X. We will make no distinctions between this set and the terms over the signature $\langle X, \cdot, \cap \rangle$ quotiented by the label space axioms.

$\mathbb{1}$ **:** the unit label space.

$\mathbb{E}_X$ **:** the set of expressions $\mathrm{Reg}^{\cap}_{\Sigma}$ (quotiented by the congruence generated by label space axioms).

3.1 Graphs

We introduce a few definitions and results about graphs. Unless otherwise stated, all graphs are labelled with some fixed label space $\mathbb{L}$.

[2] See Remark 18 for more details.

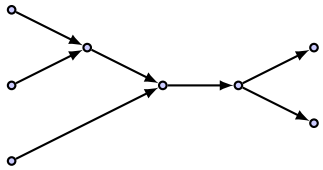


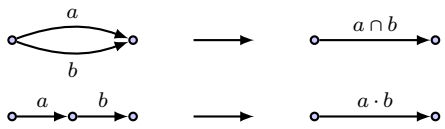
Figure 7: Example of reduced bow tie

DAGs and Series Parallel graphs Given a directed acyclic graph (DAG) $G = \langle V, E\rangle$, we may define $\min G$ to be the set of vertices in V with no incoming edge, and $\max G$ the set of vertices with no outgoing edge. A Two Terminal Parallel Series graph (SP-graph in the following) may be defined as a connected DAG that does not contain the graph from Figure 5 as a minor, and such that $\min G$ and $\max G$ are both singleton sets. A *factor* of $\langle V, E\rangle$ is a graph $\langle V', E'\rangle$ such that:

- $V' \subseteq V$ and $E' \subseteq E$;
- $e \in E'$ if and only if both its extremities are in V'.

A *prefix* of $\langle V, E\rangle$ is a factor $\langle V', E'\rangle$ such that if there is a path from $x \in V$ to $y \in V'$, then $x \in V'$. When considering the graph as a partial order, a prefix is a downward closed set of vertices. Similarly, A *suffix* of $\langle V, E\rangle$ is an upward closed set, meaning a factor $\langle V', E'\rangle$ such that if there is a path from $x \in V'$ to $y \in V$, then $y \in V'$. Finally a *slice* of $\langle V, E\rangle$ is a convex set, i.e. a factor such that if there is a path from $x \in V'$ to $y \in V$ and one from y to $z \in V'$, then $y \in V'$. Any slice may be obtained by removing from the graph a prefix and a suffix. A graph G is called an *SP-slice* if it is a slice of some SP-graph.

We define the SP-rewriting system, enriched with labels taken from a label space:



(This second rule can only be applied if the middle node on the left-hand side has no other adjacent edge.)

Given a DAG G, we write $G\downarrow$ for the unique normal form of G with respect to this system. One may notice here that if G is a non-trivial SP-graph, then $G\downarrow$ has always the shape $\to \bullet \xrightarrow{e} \bullet \to$[3], and G is isomorphic to $\mathscr{G}(e)$. This isomorphism means that we will often assimilate SP-graphs and terms built using the operators $\cdot$ and $\cap$.

Trees and bow ties A *tree* is a DAG $T = \langle V, E\rangle$ such that either $\min T$ or $\max T$ contains a single node called the root, and for any two nodes $x, y \in V$ there exists at most one path from x to y. If $\min T$ is a singleton, then T is a *top-down tree*, otherwise it is a *bottom-up tree*. A tree is said to be proper if it does not contain any node with exactly one incoming edge and one outgoing edge. It is simple enough to check that there is only a finite number of proper trees with leaves chosen from a finite set.

A *bow tie* is a DAG G that does not contain the graph from Figure 5 as a minor and such that there is a node o such that every node in G either can reach o or is reachable from o. By applying the SP-rewriting system to a bow tie, we get a bottom-up tree concatenated with a top-down tree, as illustrated in figure Figure 7 (hence the name). It follows easily that G is an SP-slice if and only if it is a disjoint union of bow ties.

[3] This is sometimes used as the definition of an SP-graph.

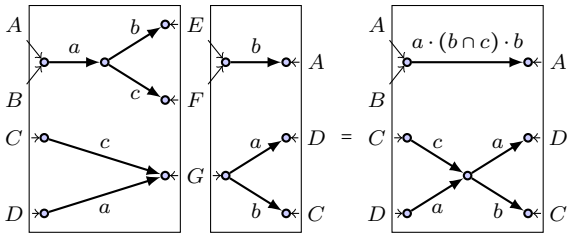


Figure 8: Composition of boxes

3.2 Categories of boxes

Definition 10 (Box)
A *box* from A to B labelled with a label space $\mathbb{L}$ is a tuple $\beta = \langle G, \underline{\mathfrak{p}}, \bar{\mathfrak{p}}\rangle$ where G is a DAG labelled with $\mathbb{L}$, $\underline{\mathfrak{p}} : A \to \min G$ is a surjective map and $\bar{\mathfrak{p}} : B \to \max G$ is a bijective map, respectively indicating the input ports and output ports. We write $\beta : A \to B$ if β is a box from A to B. ∗

Examples of such boxes are given in Figures 10 and 11.

Definition 11 (Type)
A *type* over a set A is a structure $\tau = \langle G_\tau, l_\tau\rangle$ such that G_τ is a proper top-down unlabelled tree and l_τ is a bijection from A to $\max G_\tau$. The set of types over subsets of a finite set P is written $\mathbb{T}_P$. ∗

Examples of such types are given in Figure 9.

Remark 12. In the sequel we reason about boxes and types modulo renaming of nodes.

A type τ over A may be seen as a box from $\{\iota\}$ to A labelled with $\mathbb{1}$: we can build the box from $\{\iota\}$ to A $\boxed{\tau} \coloneqq \langle V_\tau, E, [\iota \mapsto r_\tau], l_\tau\rangle$, with $E = \{(x, 1, y) \mid (x, y) \in E_\tau\}$. Conversely, we may forget about the label information in a box: the erasing of a box β is the box $\lfloor\beta\rfloor$ where all labels are replaced by 1. It is quite clear that for any type τ, $\boxed{\tau} = \lfloor\boxed{\tau}\rfloor$.

One of the most interesting features of boxes is their ability to compose. Let $\beta = \langle G_1, \underline{\mathfrak{p}}_1, \bar{\mathfrak{p}}_1\rangle : A \to B$ and $\gamma = \langle G_2, \underline{\mathfrak{p}}_2, \bar{\mathfrak{p}}_2\rangle : B \to C$. We define the relation $\sim$ over $G_1 \cup G_2$ as the smallest equivalence relation containing all pairs x, y such that there is some $b \in B$ such that $\bar{\mathfrak{p}}_1(b) = x$ and $\underline{\mathfrak{p}}_2(b) = y$. The composition of β and γ, written $\beta \odot \gamma$, is defined as $\langle G\downarrow, \underline{\mathfrak{p}}, \bar{\mathfrak{p}}\rangle : A \to C$, where:

- G is the quotient of the disjoint union of G_1 and G_2 by the relation $\sim$;
- the function $\underline{\mathfrak{p}}$ (respectively $\bar{\mathfrak{p}}$) associates to $x \in A$ (resp. $\in B$) the equivalence class of $\underline{\mathfrak{p}}_1(x)$ (resp. $\bar{\mathfrak{p}}_2(x)$).

It can be checked that this composition operation is associative and that for any two boxes $\beta : A \to B$ and $\gamma : B \to C$, the following holds:

$$\lfloor\beta \odot \gamma\rfloor = \lfloor\beta\rfloor \odot \lfloor\gamma\rfloor. \tag{1}$$

This operation is illustrated in Figure 8.

Definition 13 (Typed boxes)
Let $\beta : A \to B$ be a box over $\mathbb{L}$, σ and τ be types respectively over A and B. β *has the type* $\sigma \multimap \tau$ if the following holds:

$$\lfloor\boxed{\sigma} \odot \beta\rfloor = \boxed{\tau}. \tag{2}$$

We write $\mathfrak{B}^{\mathbb{L}}_{\sigma\multimap\tau}$ for the set of boxes over $\mathbb{L}$ of type $\sigma \multimap \tau$, and $\mathbb{B}^{\mathbb{L}}_P$ for the set of typed boxes over $\mathbb{L}$ with types from $\mathbb{T}_P$.

These boxes allow us to define the category $\mathbb{B}^{\mathbb{L}}$ of types and typed boxes over $\mathbb{L}$ where:

- the objects are types;

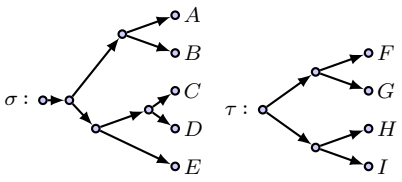


Figure 9: Some types σ and τ.

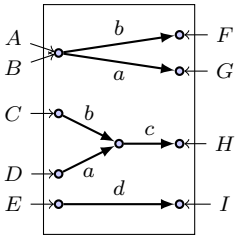


Figure 10: A box in $\mathfrak{B}^{\mathbb{W}_X}_{\sigma\multimap\tau}$.

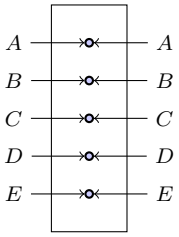


Figure 11: The identity box on type σ.

- for any two objects σ, τ, the set of morphisms from σ to τ is $\mathfrak{B}^{\mathbb{L}}_{\sigma\multimap\tau}$.

$*$

(Notice that the type of a box is not unique: a single box may have multiple types.)

For instance the box in Figure 10 has type $\sigma \multimap \tau$, with σ and τ as defined in Figure 9.

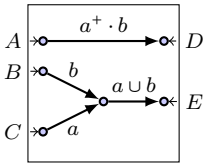
Let us verify that $\mathbb{B}^{\mathbb{L}}$ satisfies the axioms of a category. We already know that the composition of boxes is associative, but its remains to be shown that the composition of typed boxes is a typed box. In other words: $\mathfrak{B}^{\mathbb{L}}_{\sigma\multimap\tau} \odot \mathfrak{B}^{\mathbb{L}}_{\tau\multimap\theta} \subseteq \mathfrak{B}^{\mathbb{L}}_{\sigma\multimap\theta}$.
Let $\beta, \gamma \in \mathfrak{B}^{\mathbb{L}}_{\sigma\multimap\tau} \times \mathfrak{B}^{\mathbb{L}}_{\tau\multimap\theta}$. We need to check that $\beta \odot \gamma \in \mathfrak{B}^{\mathbb{L}}_{\sigma\multimap\theta}$.

$$
\begin{aligned}
\lfloor \boxed{\sigma} \odot (\beta \odot \gamma) \rfloor &= \lfloor (\boxed{\sigma} \odot \beta) \odot \gamma \rfloor && \text{(Associativity)}\\
&= \lfloor \boxed{\sigma} \odot \beta \rfloor \odot \lfloor \gamma \rfloor && \text{(Equation (1))}\\
&= \boxed{\tau} \odot \lfloor \gamma \rfloor && (\beta \in \mathfrak{B}^{\mathbb{L}}_{\sigma\multimap\tau})\\
&= \lfloor \boxed{\tau} \odot \gamma \rfloor && \text{(Equation (1))}\\
&= \boxed{\theta}. && (\gamma \in \mathfrak{B}^{\mathbb{L}}_{\tau\multimap\theta})
\end{aligned}
$$

The last thing we need is the identity over an arbitrary object τ. If L is the set of leaves of τ, then we define $\mathrm{id}_\tau = \langle \langle L, \varnothing \rangle, l_\tau, l_\tau \rangle$. It is immediate to check that $\mathrm{id}_\tau \in \mathfrak{B}^{\mathbb{L}}_{\tau\multimap\tau}$ and that for any typed boxes β, γ of appropriate types, $\beta \odot \mathrm{id}_\tau = \beta$ and $\mathrm{id}_\tau \odot \gamma = \gamma$. The identity box on type τ is displayed in Figure 11.

3.3 Boxes over expressions

Consider boxes labelled with a label space $\mathbb{E}_X$. The elements of $\mathbb{E}_X$ are expressions taken from $\mathrm{Reg}^{\cap}_X$, and thus denote languages of SP-graphs, as defined earlier. This means that we can generate from such a box a set of boxes labelled with $\mathbb{W}_X$, by replacing each edge labelled with e by a graph $G \in \mathscr{G}(e) \subseteq \mathbb{W}_X$[4]. For instance, the box:



[4] As mentioned before, we make no distinction between SP-graphs and terms built using letters and the operators $\cdot$ and $\cap$.

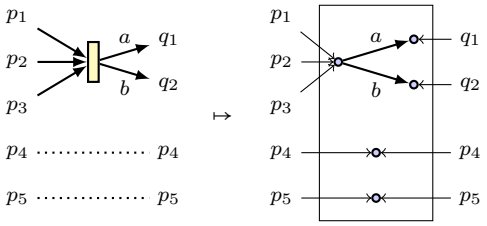
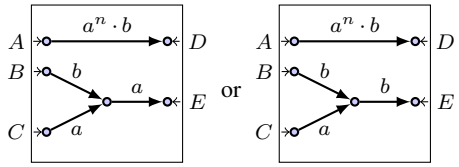


Figure 12: Construction of $\beta_{t,C}$

can generate all boxes of the following shapes, for $n > 0$:



It is useful to notice that for any box $B \in \mathfrak{B}^{\mathbb{E}_X}_{\sigma\multimap\tau}$, if B generates the box β, then $\beta \in \mathfrak{B}^{\mathbb{W}_X}_{\sigma\multimap\tau}$. In the following, we will denote by $\mathfrak{B}(B)$ the set of boxes generated by a box B labelled with $\mathbb{E}_X$.

4. Automata and regular sets of boxes

4.1 A regular expression over boxes

It is then quite easy to generate from an SP-automaton $\mathscr{A}$ a regular expression over boxes $e_{\mathscr{A}} \in \mathrm{Reg}\langle \mathbb{B}_P^{\mathbb{W}_\Sigma \bullet} \rangle$ such that:

$$
\beta_1 \cdots \beta_n \in [\![e_{\mathscr{A}}]\!] \Leftrightarrow \begin{cases} \beta_1 \odot \cdots \odot \beta_n = \langle G, \underline{\mathfrak{p}}, \bar{\mathfrak{p}} \rangle \text{ with } G \in \mathscr{G}(\mathscr{A}) \\ \beta_1 \odot \cdots \odot \beta_n \text{ has type } \bullet \longrightarrow \bullet \; \iota \multimap \; \bullet \longrightarrow \bullet \; f \ . \end{cases}
$$

Let us fix an SP-automaton $\mathscr{A} = \langle P, \mathscr{T}, \iota, f \rangle$ over an alphabet Σ. Its transitions are thus labelled with $\Sigma_\bullet$. Given automata configurations C and C', and a transition t such that $C \xrightarrow{t}_{\mathscr{A}} C'$, we can define a box $\beta_{t,C} : C \to C'$ as follows. Suppose $\overline{t} = \{(a_1, q_1), \ldots, (a_m, q_m)\}$ and $C \setminus \underline{t} = \{p_1, \ldots, p_n\}$. Recall that because t is a valid transition from C to C', the sets $\{q_1, \ldots, q_m\}$ and $\{p_1, \ldots, p_n\}$ are disjoint and $C' = \{p_1, \ldots, p_n, q_1, \ldots, q_m\}$. Thus $\beta_{t,C}$ is defined as $\langle V, E, \underline{\mathfrak{p}}, \bar{\mathfrak{p}} \rangle$ where:

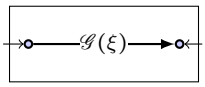
- V contains an is the input node x_0, a node x_i for every place p_i in $C \setminus \underline{t}$ and a node y_i for every output q_i.
- $\underline{\mathfrak{p}}$ maps every input place p to x_0, every every other p_i in $C \setminus \underline{t}$ to x_i.
- $\bar{\mathfrak{p}}$ maps every output q_i to y_i, and the remaining p_i to x_i.
- for each output (a_i, q_i), we place an edge (x_0, a_i, y_i) in E.

This construction is illustrated in Figure 12.

We may extend this construction to runs: for every run of length n $\xi = \langle (\xi_k)_{0\leqslant k\leqslant n}, (t_k)_{0\leqslant k<n} \rangle$ we define $\beta(\xi)$ by:

$$
\beta(\xi) \coloneqq \beta_{t_0,\xi_0} \odot \cdots \odot \beta_{t_{n-1},\xi_{n-1}}.
$$

It is quite straight-forward to check that for every accepting run ξ,



Thus there is no difference between the graphs produced by the automaton and the boxes corresponding to its accepting runs. We call a box *valid* if there is an accepting run ξ with a consecutive sub-run $\xi[i..j]$ such that $\beta = \beta(\xi[i..j])$. We say in that case that the run $\xi[i..j]$ *witnesses* the validity of β. Notice that because $\mathscr{A}$ is an SP-automaton, the graph of a valid box has to be an SP-slice.

The types help us here, as illustrated by the following lemma:

Lemma 14. *If β is a valid box with type $\sigma \multimap \tau$ and γ is valid with type $\tau \multimap \upsilon$, then $\beta \odot \gamma$ is a valid box with type $\sigma \multimap \gamma$.*

We extend this to words and regular expressions over typed boxes.

Definition 15 (Valid expression)
An expression $e \in \mathrm{Reg}\langle \mathfrak{B}^{\mathbb{W}_{\Sigma\bullet}} \rangle$ is valid if for all $\beta_1 \cdots \beta_n \in [\![e]\!]$, the composition $\beta_1 \odot \cdots \odot \beta_n$ is well defined, and the resulting box is valid. $*$

Given a word $w = \beta_1 \cdots \beta_n$, we call the box $\beta_1 \odot \cdots \odot \beta_n$ the *compilation* of w.

Definition 16 (Box language of e)
If $e \in \mathrm{Reg}\langle \mathfrak{B}^{\mathbb{W}_{\Sigma\bullet}} \rangle$ is valid, then the box language of e, written $[\![e]\!]_\odot$, is the set of boxes obtained by compiling the words of $[\![e]\!]$. $*$

Definition 17 (Typed expression)
A valid expression $e \in \mathrm{Reg}\langle \mathbb{B}_P^{\mathbb{W}_{\Sigma\bullet}} \rangle$ has type $\sigma \multimap \tau$ if for all $\beta \in [\![e]\!]_\odot$, β has the type $\sigma \multimap \tau$. $*$

We may now define a deterministic finite-state automaton as follows:

- the set of states is $\mathbb{T}_P$, the types over the set P of places of $\mathscr{A}$ (as remarked before, this set is finite);
- for every pair of types σ, τ, if C is the set of leaves of σ, if $t \in \mathscr{T}$ can be fired from C and if $\beta_{t,C} \in \mathfrak{B}^{\mathbb{W}_{\Sigma\bullet}}_{\sigma \multimap \tau}$ we add a transition $(\sigma, \beta_{t,C}, \tau)$;
- the initial state is the type $\circ\!\!\longrightarrow\!\!\circ \iota$;
- the final state is the type $\circ\!\!\longrightarrow\!\!\circ f$.

Remark 18. This automaton can be computed partially, by enumerating all runs starting from $\{\iota\}$, and computing the corresponding boxes and types. Because of the finite number of states in this automaton this procedure must terminate. If the automaton we were given is not SP, then the computation will reach a step where we have a reachable type σ with leaves C and a transition t enabled at C such that $\lfloor \boxed{\sigma} \odot \beta_{t,C} \rfloor$ does not reduce to a tree. This gives a decision procedure to check whether $\mathscr{A}$ is indeed an SP-automaton.

Using the classic Kleene theorem, we can then compile this automaton into a regular expression $e_{\mathscr{A}} \in \mathrm{Reg}\langle \mathbb{B}_P^{\mathbb{W}_{\Sigma\bullet}} \rangle$ such that:

1. $e_{\mathscr{A}}$ and all its sub-expressions are valid and typed, as defined earlier, meaning that the words in their languages compile to valid boxes and that two boxes in the language of the same sub-expression share a common type[5];
2. if ξ is an accepting run of the Petri automaton $\mathscr{A}$, then $\beta(\xi)$ is the compilation of some word in $[\![e]\!]_{\mathscr{A}}$. Thus $[\![e_{\mathscr{A}}]\!]_\odot$ is the set of all valid boxes of type $\circ\!\!\longrightarrow\!\!\circ \iota \multimap \circ\!\!\longrightarrow\!\!\circ f$.

This allow us to state an important property of the expression $e_{\mathscr{A}}$:

Lemma 19. *Let $f, g \in Reg\langle \mathbb{B}_P^{\mathbb{W}_{\Sigma\bullet}} \rangle$ be valid expressions of type $\sigma \multimap \tau$, such that $[\![f]\!] \subseteq [\![g]\!]$. If $e_{\mathscr{A}}[f/g]$ is the expression obtained by replacing an occurrence of f with g in $e_{\mathscr{A}}$, then the box language of $e_{\mathscr{A}}$ is equal to that of $e_{\mathscr{A}}[f/g]$.*

Proof. We proceed by mutual inclusion. One direction is straight forward: all regular operations being increasing, we know that $[\![f]\!] \subseteq [\![g]\!]$ entails $[\![e_{\mathscr{A}}]\!] \subseteq [\![e_{\mathscr{A}}[f/g]]\!]$, thus ensuring $[\![e_{\mathscr{A}}]\!]_\odot \subseteq [\![e_{\mathscr{A}}[f/g]]\!]_\odot$.

[5] This property follows from results about typed Kleene Algebra. See for instance [4, 5, 7].

The property 2 allows us to check the other direction. The typing ensures that for any words u, v, w and w', if w and w' compile to boxes with the same type then $u \cdot w \cdot v$ is valid if and only if $u \cdot w' \cdot v$ is valid as well, and they have the same type. Thus we know that $[\![e_{\mathscr{A}}[f/g]]\!]_\odot$ only contain valid boxes of the same type as the ones from $[\![e_{\mathscr{A}}]\!]_\odot$. As $[\![e_{\mathscr{A}}]\!]_\odot$ is maximal for its type, it must contain $[\![e_{\mathscr{A}}[f/g]]\!]_\odot$. □

4.2 Compiling an expression over boxes into $\mathrm{Reg}^{\cap}_{\Sigma}$

What remains to do is to transform this regular expression over boxes into an expression from $\mathrm{Reg}^{\cap}_{\Sigma}$. We do so by moving from typed boxes labelled with $\mathbb{W}_{\Sigma\bullet}$ to finite sets of boxes sharing a common type and labelled with $\mathbb{E}_{\Sigma\bullet} \supseteq \mathbb{W}_{\Sigma\bullet}$, thus internalising the operators. By mapping each box β to $\{\beta\}$, we may translate this way the expression $e_{\mathscr{A}}$ obtained in the previous section to a new expression $\mathfrak{e}_{\mathscr{A}}$. The operator $\odot$ may be lifted here in a type preserving way, as a pointwise application:

$$S \odot S' \coloneqq \left\{ B \odot B' \;\middle|\; B \in S, B' \in S' \right\}.$$

We also extend the notion of boxes over $\mathbb{W}_{\Sigma\bullet}$ generated by a box over $\mathbb{E}_{\Sigma\bullet}$ to a set S simply by stating that $\mathfrak{B}(S) \coloneqq \bigcup_{B \in S} \mathfrak{B}(B)$.

Definition 20 (Valid expression)
An expression $e \in \mathrm{Reg}\langle \mathcal{P}_f\left(\mathbb{B}_P^{\mathbb{E}_{\Sigma\bullet}}\right)\rangle$ is valid if all $S_1 \cdots S_n \in [\![e]\!]$, the composition $S_1 \odot \cdots \odot S_n = S$ is defined and $\mathfrak{B}(S)$ only contains valid boxes. $*$

The definition of $[\![\_]\!]_\odot$ we had for valid expressions with letters in $\mathbb{B}_P^{\mathbb{W}_{\Sigma\bullet}}$ can thus be salvaged for expressions with letters in $\mathcal{P}_f\left(\mathbb{B}_P^{\mathbb{E}_{\Sigma\bullet}}\right)$. Lemma 14 can be adapted to this setting:

Lemma 21. *Let S_1 and S_2 be finite sets of boxes over $\mathbb{E}_{\Sigma\bullet}$, such that*

1. *$\forall \beta \in \mathfrak{B}(S)$, β is a valid box with type $\sigma \multimap \tau$;*
2. *$\forall \gamma \in \mathfrak{B}(S')$, γ is a valid box with type $\tau \multimap \upsilon$;*

then $\forall \beta \in \mathfrak{B}(S \odot S')$, β is a valid box with type $\sigma \multimap \gamma$.

A regular expression e with letters in $\mathcal{P}_f\left(\mathbb{B}_P^{\mathbb{E}_{\Sigma\bullet}}\right)$ represent the set of boxes

$$(\!|e|\!) \coloneqq \bigcup_{S \in [\![e]\!]_\odot} \mathfrak{B}(S).$$

We extend a definition from the previous section to this setting:

Definition 22 (Typed expression)
An expression valid $e \in \mathrm{Reg}\langle \mathcal{P}_f\left(\mathbb{B}_P^{\mathbb{E}_{\Sigma\bullet}}\right)\rangle$ has type $\sigma \multimap \tau$ if all $\beta \in (\!|e|\!)$ have the type $\sigma \multimap \tau$. $*$

We can now equate two expressions if they denote the same set of boxes, introducing the equivalence relation $\equiv_\beta$:

$$e \equiv_\beta f \coloneqq ((\!|e|\!) = (\!|f|\!))$$

Notice that Lemma 19 can be restated for $\mathfrak{e}_{\mathscr{A}}$:

Lemma 23. *Let $f, g \in Reg\langle \mathcal{P}_f\left(\mathfrak{B}^{\mathbb{E}_{\Sigma\bullet}}\right)\rangle$ be valid expressions of type $\sigma \multimap \tau$, such that $[\![f]\!]_\odot \subseteq [\![g]\!]_\odot$. Then $\mathfrak{e}_{\mathscr{A}} \equiv_\beta \mathfrak{e}_{\mathscr{A}}[g/f]$.*

The operator $\cup$ and $\cdot$ are easy enough to define on finite sets of boxes labelled over $\mathbb{E}_{\Sigma\bullet}$:

$$S \cdot S' \equiv_\beta \bigcup_{B \in S} \bigcup_{B' \in S'} B \odot B'$$

$$S \cup S' \equiv_\beta \left\{ B \;\middle|\; B \in S \cup S' \right\}$$

The case of $\_^\star$ however is much more interesting. In fact, this is the main technical difficulty of the present paper, and where most of the machinery we have introduced up to this point will prove

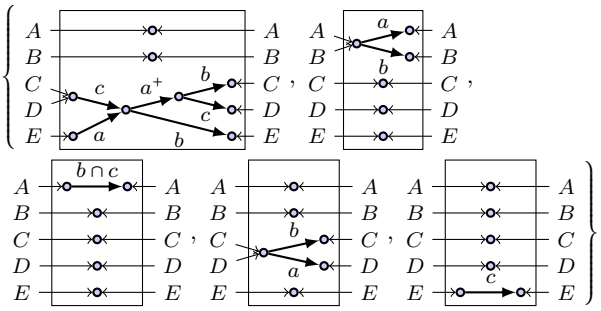


Figure 13: Elementary decomposition of $\{\beta, \gamma\}$.

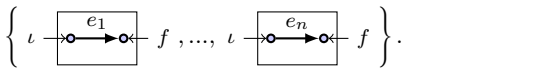
useful. For typing reasons, notice that we only need to compute the star of a set of boxes of an identity type $\sigma \multimap \sigma$. In the next section, we show how to compute from a set S of boxes labelled over $\mathbb{E}_{\Sigma^\bullet}$ and having type $\sigma \multimap \sigma$ a new finite set of boxes S' such that:

1. $(\!|S^\star|\!) \subseteq (\!|S'|\!)$
2. and $\forall \beta \in (\!|S'|\!)$, β is valid and has type $\sigma \multimap \sigma$.

By Lemma 23 this entails the following equivalence:

$$\mathfrak{e}_{\mathscr{A}} \equiv_\beta \mathfrak{e}_{\mathscr{A}}\left[S'/\left(S^\star\right)\right].$$

Using these transformation, we now can "compile" the expression $\mathfrak{e}_{\mathscr{A}}$ into a finite set of boxes of type $\bullet\!\!\rightarrow\!\!\bullet\ \iota \multimap \bullet\!\!\rightarrow\!\!\bullet\ f$. Such a set may only have the shape:



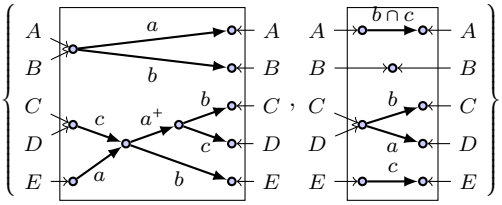
It is then straightforward to see that $(\!|\mathfrak{e}_{\mathscr{A}}|\!)$ is isomorphic to $\mathscr{G}(e_1 \cup \cdots \cup e_n)$. If we call $\mathscr{E}(\mathscr{A})$ the expression $e_1 \cup \cdots \cup e_n$, the following theorem holds:

Theorem 24 (Main result)**.** *For every SP-automaton $\mathscr{A}$ over Σ, there exists an expression $\mathscr{E}(\mathscr{A}) \in Reg^{\cap}_{\Sigma}$ such that*

$$\mathscr{G}(\mathscr{A}) = \mathscr{G}(\mathscr{E}(\mathscr{A})).$$

4.3 Staring boxes

Instead of giving a formal description, which would be very involved, we describe the procedure on an example, outlining the important computation steps. Consider the following set of boxes, respectively named β and γ.



Both these boxes have type $\sigma \multimap \sigma$, with the σ as shown in Figure 9. First, we decompose each box, by separating disjoint non-trivial connected components. The idea behind this step is that the different connected component correspond to concurrent processes, and as such may be iterated independently. This yields the set of five so-called *elementary* boxes displayed in Figure 13, respectively named η_1 through η_5.

One can show that the resulting boxes are always valid and have the type $\sigma \multimap \sigma$(a formal proof of the properties of this decomposition is given in Appendix A). Because $\beta = \eta_1 \odot \eta_2$ and $\gamma = \eta_3 \odot \eta_4 \odot \eta_5$, we get that:

$$[\![(\beta \cup \gamma)^\star]\!]_\odot \subseteq [\![(\eta_1 \cup \eta_2 \cup \eta_3 \cup \eta_4 \cup \eta_5)^\star]\!]_\odot$$

box	support
η_1	$\{C, D, E\}$
η_2	$\{A, B\}$
η_3	$\{A\}$
η_4	$\{C, D\}$
η_5	$\{E\}$

Figure 14: Supports of the η_i

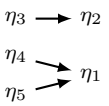


Figure 15: Graph of $\preccurlyeq$

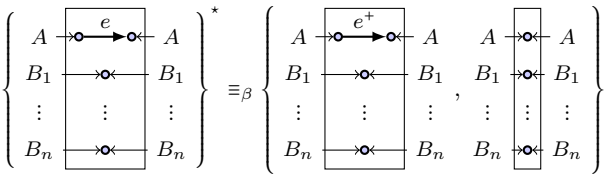
On the other hand, because of Lemma 21 we can establish that the set $(\!|(\eta_1 \cup \eta_2 \cup \eta_3 \cup \eta_4 \cup \eta_5)^\star|\!)$ only contains valid boxes of type $\sigma \multimap \sigma$.

By Lemma 23, we know that we may replace in the expression $\mathfrak{e}_{\mathscr{A}}$ the sub expression $(\beta \cup \gamma)^\star$ by $(\eta_1 \cup \eta_2 \cup \eta_3 \cup \eta_4 \cup \eta_5)^\star$ without changing the set of produced graphs. This means that we may continue the computation with the set of elementary boxes $H = \{\eta_1, \eta_2, \eta_3, \eta_4, \eta_5\}$. We define the *support* of a box with a single non-trivial connected component as the set of ports that are mapped inside this component. Figure 14 lists the supports of the boxes η_i. Notice that for every two boxes η, η' in H, their supports are either included one in the other, or of empty intersection. This can be shown formally by realising that for a box with a single connected component of type $\sigma \multimap \sigma$, the support is the set of leaves of a sub-tree of σ (i.e. the set of all leaves reachable from some node of σ). This means we can endow the set $\{\eta_1, \eta_2, \eta_3, \eta_4, \eta_5\}$ with a pre-order relation $\preccurlyeq$ defined by $\eta \preccurlyeq \eta'$ if $support(\eta) \subseteq support(\eta')$. The graph of the order on our example set is given in Figure 15.

The order has an interesting property: if η and η' are incomparable, then $\eta \odot \eta' = \eta' \odot \eta$. Thus $(\eta \cup \eta')^\star \equiv_\beta \eta^\star \cdot \eta'^\star$. With that in mind we will compute the expression:

$$\left(\eta_3^\star \cup \eta_2\right)^\star \cdot \left(\left(\eta_4^\star \cdot \eta_5^\star\right) \cup \eta_1\right)^\star.$$

The first step in this computation is the computation of the star of a single box. Notice that we may define easily the star of a "linear" elementary box, i.e. of the shape:



This construction gives us $\eta_3^\star$ and $\eta_5^\star$. Now for general elementary boxes, notice that the non-trivial connected component of an elementary box is always a bow tie. Hence we can always decompose such a box η of type $\sigma \multimap \sigma$ as $\delta_1 \odot \delta_2$ such that the non-trivial connected component of δ_1 is a bottom-up tree, and that of δ_2 is a top-down tree, with the middle node o of the bow tie being the root of both trees. See Figure 16 for examples. This means that $\delta_2 \odot \delta_1$ is a linear elementary box. Furthermore, the definition of $\delta_{1,2}$ ensures $\{\eta\}^\star \equiv_\beta (\{\delta_1\} \cdot \{\delta_2\})^\star$. The regular laws entail that for every expressions e and f we have $[\![(e \cdot f)^\star]\!] = [\![1 \cup e \cdot (f \cdot e)^\star \cdot f]\!]$. Thus we can use the previous construction to get a set of boxes representing $(\{\delta_2\} \cdot \{\delta_1\})^\star \equiv_\beta \{\delta_2 \odot \delta_1\}^\star$, and then compute:

$$\{\eta\}^\star \equiv_\beta \{\mathrm{id}_\sigma\} \cup \{\delta_1\} \cdot \{\delta_2 \odot \delta_1\}^\star \cdot \{\delta_2\}.$$

The last thing to do is to compute stars of the shape $(S^\star \cup \eta)^\star$ where:

- $S^\star$ is a finite set of boxes computed in an earlier step,
- $support(\eta)$ contains the "support" of $S^\star$: the set of inputs mapped to a non trivial connected component in some $B \in S^\star$.

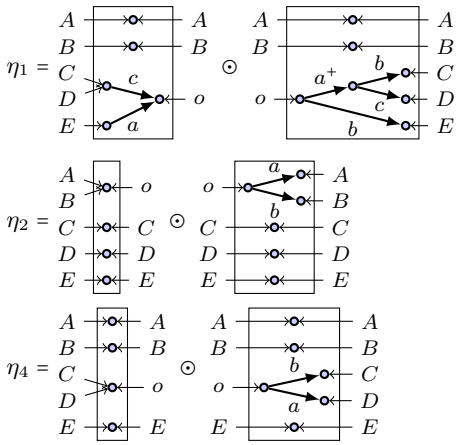
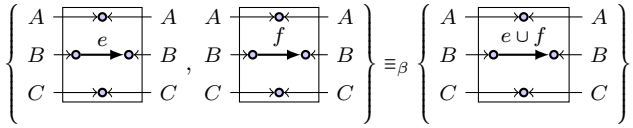


Figure 16: Decomposition as a product of trees of η_1, η_2 and η_4.

Again, we decompose η into δ_1 and δ_2 as in the previous construction, and notice that $\delta_2 \odot S^\star \odot \delta_1$ is set of a linear boxes, along the same line. We may thus bundle them together in a single box with a line labelled with the union of their labels. For example:



This allow us to produce a set of (two) boxes representing the star of the box $(\delta_2 \odot S^\star \odot \delta_1)$. The regular laws allow us to conclude, by stating that:

$$\left(S^\star \cup \delta_1 \cdot \delta_2\right)^\star \equiv_\beta S^\star \cup \delta_1 \odot \left(\delta_2 \odot S^\star \odot \delta_1\right)^\star \odot \delta_2$$

We can now give the bulk of the computation, in Figure 17. The result is then obtained by performing the composition

$$\left(\eta_3^\star \cup \eta_2\right)^\star \odot \left(\left(\eta_4^\star \cup \eta_5\right)^\star \cup \eta_1\right)^\star,$$

thus yielding 32 boxes: 8 for $\left(\left(\eta_4^\star \cup \eta_5^\star\right) \cup \eta_1\right)^\star$ times 4 for $\left(\eta_4^\star \cup \eta_5\right)^\star$.

5. Consequences

5.1 Kleene Allegories

The main motivation for this work was the study of the equational theory of Kleene Allegories (KAl). In this line of work we consider binary relations and the operations of union ($\cup$), intersection ($\cap$), composition ($\cdot$), converse ($\_^\smile$), transitive closure ($\_^+$), and the constants identity (1) and empty relation (0). Terms made from those operations and some variables $a, b, \cdots \in \Sigma$ constitute the set $\mathrm{Reg}_\Sigma^{\smile\cap}$. A pair $e, f \in \mathrm{Reg}_\Sigma^{\smile\cap}$ is then *relationaly equivalent*, denoted $\mathrm{Rel} \vDash e = f$, if the corresponding equality holds universally when the expressions are interpreted as binary relations. Notice that we may restrict expressions so that the operator $\_^\smile$ is only applied to variables, by using the following rewriting system:

$$\begin{array}{lll} (a \cup b)^\smile \to a^\smile \cup b^\smile & 0^\smile \to 0 & (a^+)^\smile \to (a^\smile)^+ \\ (a \cdot b)^\smile \to b^\smile \cdot a^\smile & 1^\smile \to 1 & a^{\smile\smile} \to a \\ (a \cap b)^\smile \to a^\smile \cap b^\smile. & & \end{array}$$

The result of this transformation on an expression e is an expression e' always satisfying $\mathrm{Rel} \vDash e = e'$. We may associate to these terms graph languages, as in Definition 2.

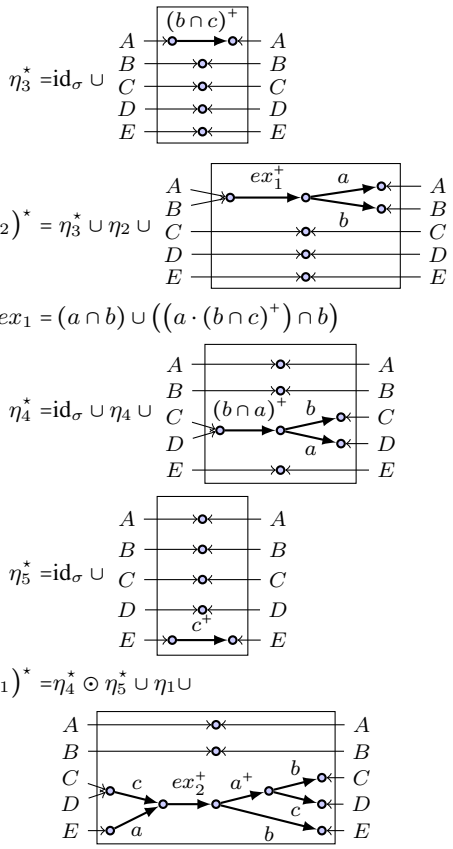


Figure 17: Computation steps

Definition 25 (Graph language of a KAl expression: $\mathscr{G}'(e)$)
The graph language is defined by structural induction as follows:

$$\mathscr{G}'(a) \coloneqq \left\{ \xrightarrow{\ a\ } \right\}; \quad \mathscr{G}'(a^\smile) \coloneqq \left\{ \xleftarrow{\ a\ } \right\};$$
$$\mathscr{G}'(0) \coloneqq \varnothing; \quad \mathscr{G}'(1) \coloneqq \{ \longrightarrow \};$$
$$\mathscr{G}'(e \cup f) \coloneqq \mathscr{G}'(e) \cup \mathscr{G}'(f);$$
$$\mathscr{G}'(e \cdot f) \coloneqq \left\{ E \cdot F \;\middle|\; E \in \mathscr{G}'(e) \text{ and } F \in \mathscr{G}'(f) \right\};$$
$$\mathscr{G}'(e \cap f) \coloneqq \left\{ E \cap F \;\middle|\; E \in \mathscr{G}'(e) \text{ and } F \in \mathscr{G}'(f) \right\};$$
$$\mathscr{G}'\left(e^+\right) \coloneqq \left\{ E_1 \cdot \ldots \cdot E_n \;\middle|\; n > 0,\ \forall i, E_i \in \mathscr{G}'(e) \right\}.$$

*

In [2], we showed the following result:

$$\mathrm{Rel} \vDash e = f \Leftrightarrow {}^\blacktriangleleft\mathscr{G}'(e) = {}^\blacktriangleleft\mathscr{G}'(f).^{6}$$

[6] We need not concern ourselves with the definition of ${}^\blacktriangleleft\_$ here, as it plays no role in the present result.

Notice that the only two differences between $\mathscr{G}(\_)$ and $\mathscr{G}'(\_)$ are the construction for 1 and the case of $\_^{\smile}$. We can mend that gap, by computing $\mathscr{G}'(e)$ from $\mathscr{G}(e)$. The first step is to consider the sub-expressions $a^{\smile}$ as letters, i.e. to see $e \in \mathrm{Reg}_{\Sigma}^{\smile\cap}$ as an expression in $\mathrm{Reg}_{\mathfrak{S}}^{\cap}$ with $\mathfrak{S} \coloneqq \Sigma \cup \{a^{\smile} \mid a \in \Sigma\}$. This means we can apply $\mathscr{G}(\_)$ on e, and produce a set of graphs labelled with $\mathfrak{S}$. We then introduce the graph transformation Φ:

Definition 26 (Φ)
Let $G = \langle V, E, \iota, o\rangle$ be a graph labelled with $\mathfrak{S}$. Let $\equiv_G$ be the smallest equivalence relation on V containing all pairs (i,j) such that $(i,1,j) \in E$. Then $\Phi(G)$ is the graph defined by

$$\Phi(G) \coloneqq \langle \{[i]_G \mid i \in V\}, E', [\iota]_G, [o]_G\rangle$$
$$[i]_G \coloneqq \{k \in V \mid i \equiv_G k\}$$
$$E' \coloneqq \left\{ ([i]_G, x, [j]_G) \;\middle|\; \begin{array}{l} x \in \Sigma \text{ and} \\ \exists k \in [i]_G, l \in [j]_G : \\ \quad (k,x,l) \in E \text{ or} \\ \quad (l, x^{\smile}, k) \in E \end{array} \right\} \qquad *$$

It is then a simple matter to check that

$$\forall e \in \mathrm{Reg}_{\Sigma}^{\smile\cap},\ \mathscr{G}'(e) = \{\Phi(G) \mid G \in \mathscr{G}(G)\}.$$

In the same paper we also introduced Petri automata, but again we used a different definition of the language $\mathscr{L}(\mathscr{A})$ of an automaton $\mathscr{A}$. However, this set can be expressed with our notations as $\mathscr{L}(\mathscr{A}) = {}^{\blacktriangleleft}\Phi(\mathscr{G}(\mathscr{A}))$. A function $\mathscr{A}(\_)$ was provided, mapping an expression $e \in \mathrm{Reg}_{\Sigma}^{\smile\cap}$ to a Petri automaton $\mathscr{A}(e)$ such that $\mathscr{G}(\mathscr{A}(e)) = \mathscr{G}(e)$.[7] Thus:

$$\mathscr{L}(\mathscr{A}(e)) = {}^{\blacktriangleleft}\Phi(\mathscr{G}(e)).$$

Theorem 24 allow us to relate the language of any SP-automaton $\mathscr{A}$ to the graph language of $\mathscr{E}(\mathscr{A})$:

$$\mathscr{L}(\mathscr{A}) = {}^{\blacktriangleleft}\Phi(\mathscr{G}(\mathscr{E}(\mathscr{A}))).$$

This proves that from a decidability stand point, language equivalence of Petri automata and relational equivalence of KAl expressions. Formally:

$$\forall e, f,\ \mathrm{Rel} \vDash e = f \Leftrightarrow \mathscr{L}(\mathscr{A}(e)) = \mathscr{L}(\mathscr{A}(f)) \qquad (3)$$
$$\forall \mathscr{A}, \mathscr{B},\ \mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{B}) \Leftrightarrow \mathrm{Rel} \vDash \mathscr{E}(\mathscr{A}) = \mathscr{E}(\mathscr{B}) \qquad (4)$$

One could also read this result from a graph theoretical viewpoint. What we show here is that the sets of SP-graphs one may express trough regular expressions enhanced with a parallel operator $\cap$ are exactly those generated by SP-automata.

5.2 Petri nets

The construction presented here can also provide a finite regular expression with intersection to represent all finite occurrence nets of a given Petri net N, provided this net is safe and Parallel Series. In terms of concurrent processes, these two conditions roughly amount to requiring that there is a bounded number of processes running at any given time, and that communication between processes is synchronous.

More precisely, suppose we are given a net N, with a finite set P of places and a finite set $T \subseteq \mathcal{P}(P) \times \mathcal{P}(\Sigma \times P)$ of transitions.[8] If $I, F \subseteq \mathcal{P}(P)$ are respectively a source set and a target set, we can produce a Petri automaton $\mathscr{A}$ such that the set of occurrence nets of process going from a source configuration in I to a target configuration in F is isomorphic to $\mathscr{G}(\mathscr{A})$. To achieve this, we add to the net:

- two new places ι and f and a fresh label $\#$;
- for each source configuration $\{p_1, ..., p_n\} \in I$ a transition $\langle \{\iota\}, \{\langle \#, p_1\rangle, ..., \langle \#, p_n\rangle\}\rangle$;
- for each target configuration $\{q_1, ..., q_n\} \in F$ a transition $\langle \{q_1, ..., q_n\}, \{\langle \#, f\rangle\}\rangle$.

It is then a simple matter of unfolding definitions to see that $\mathscr{G}(\mathscr{A})$ indeed corresponds to the set of occurrence nets we wanted. Of course, this construction is ill-defined if the net N is not safe for one of the source configurations in I.

If furthermore the produced automaton is SP, which is again a property of the net N, then the construction of this paper yields an expression $\mathscr{E}(\mathscr{A})$ representing faithfully the set of occurrence nets of processes going from any configuration in I to any configuration in F.

[7] The automaton $\mathscr{A}(e)$ always satisfied the Parallel Series condition we required in Section 2.3.

[8] Here we stick to the setting of this paper by labelling the outputs of transitions rather than the transitions themselves. However, one can start with a net with usual labelled transition, and then simply copy the label of t to each of its outputs. This would lead to graphs where each node would have the same label on each outgoing edge, thus effectively labelling the node rather than the edge.

References

[1] H. Andréka and D. Bredikhin. The equational theory of union-free algebras of relations. *Alg. Univ.*, 33(4):516–532, 1995.

[2] P. Brunet and D. Pous. Petri automata for Kleene allegories. In *Proc. LICS*, 2015. to appear.

[3] P. J. Freyd and A. Scedrov. *Categories, Allegories*. NH, 1990.

[4] D. Kozen. Typed kleene algebra. Technical report, Cornell University, 1998.

[5] B. Möller. Typed kleene algebras. In *Mathematics Of Program Construction, Volume 3125 of LNCS*. Citeseer, 1999.

[6] T. Murata. Petri nets: Properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–580, Apr 1989.

[7] D. Pous. Untyping typed algebras and colouring cyclic linear logic. *Logical Methods in Computer Science*, 8(2), 2012.

[8] J. Valdes, R. E. Tarjan, and E. L. Lawler. The recognition of series parallel digraphs. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pages 1–12, New York, NY, USA, 1979. ACM.

A. Omitted proofs

Definition 27 (Connected component)
A set C of nodes of a graph $G = \langle V, E\rangle$ is a *connected component* if $\forall x \in C, \forall y \in V$, y is in C if and only if there is a non-oriented path from x to y in G. $*$

This means any two nodes in C are linked by non-oriented path, and that the set C is maximal for this property (with the containment order).

Definition 28 (Elementary boxes)
A box η is said to be *elementary* if its graph contains a single non-trivial connected component. $*$

Lemma 29. *If η is a valid elementary box, then its graph is necessarily a bow tie.*

Proof. Validity means the graph has to be an SP-slice, and reduced connected SP-slices are always bow ties. □

Definition 30 (Support and target of an elementary box)
Let η be an elementary box. The *support* (respectively *target*) of η is the set of input (resp. output) ports of η which are mapped inside its unique non-trivial connected component. $*$

Lemma 31. *Any valid box β can be decomposed into a set of valid elementary boxes $\eta_1, .., \eta_p$ such that:*

1. $\beta = \eta_1 \odot \cdots \odot \eta_p$*;*
2. $\forall i \neq j, support(\eta_i) \cap support(\eta_j) = \varnothing$*;*
3. *if β has type $\sigma \multimap \sigma$, then each η_i has the same type and $support(\eta_i) = target(\eta_i)$.*

Proof. We number the connected components of β from C_1 to C_p. Suppose β is valid, then there is a run $\xi = \xi_0 \xrightarrow{t_1} \cdots \xrightarrow{t_n} \xi_n$ such that $\beta = \beta(\xi) = \beta_{t_1,\xi_0} \odot \cdots \odot \beta_{t_n,\xi_{n-1}}$. Notice that for any index i, the box $\beta_{t_i,\xi_{i-1}}$ is elementary (see the definition of $\beta_{t,C}$). Thus it can be associated with a single non-trivial connected component in β. We introduce the function $c : \{1, ..., n\} \to \{1, ..., p\}$, such that the transition i is associated with the component $C_{c(i)}$.

The key argument is the following. For any index i, if the set of output places of transition t_i contains a place that appears in the input of the transition t_{i+1}, then the box $\beta_{t_i,\xi_{i-1}} \odot \beta_{t_{i+1},\xi_i}$ is elementary. This means that $c(i) = c(i+1)$. By contraposition, we may then deduce that if $c(i) \neq c(i+1)$, then the intersection of the outputs of t_i and the inputs of t_{i+1} is empty. This can in turn be used to build a configuration ξ'_i such that:

- $\xi_{i-1} \xrightarrow{t_{i+1}} \xi'_i \xrightarrow{t_i} \xi_{i+1}$;
- and $\beta_{t_i,\xi_{i-1}} \odot \beta_{t_{i+1},\xi_i} = \beta_{t_{i+1},\xi_{i-1}} \odot \beta_{t_i,\xi'_i}$.
  $(\xi' = \xi_{i-1} \setminus \underline{t_{i+1}} \cup \left\{p \;\middle|\; \exists a : (a,p) \in \overline{t_{i+1}}\right\})$

This means we may group the transitions that contribute to the same connected component. Formally we apply a permutation π to the run ξ, such that:

- $\pi(\xi) = \xi_0 \xrightarrow{t_{\pi(1)}} \cdots \xrightarrow{t_{\pi(n)}} \xi_n$
- $\beta(\xi) = \beta(\pi(\xi))$;
- $\pi(\xi)$ can be decomposed as a sequence of runs:

$$\pi(\xi) = \xi_0 \xrightarrow{\chi_1} \xi'_1 \cdots \xi'_{p-1} \xrightarrow{\chi_p} \xi_n,$$

such that $t_i \in \chi_j \Leftrightarrow c(i) = j$.

We now define $\eta_j \coloneqq \beta(\chi_j)$. By construction η_j is valid, it contains only the connected component C_j so it is elementary, and $\beta = \beta(\xi) = \beta(\pi(\xi)) = \eta_1 \odot \cdots \odot \eta_p$. Furthermore, for each $p \in \xi_0$, $p \in support(\eta_j)$ if and only if $\underline{\mathfrak{p}}(p) \in C_j$. As the connected components are pairwise disjoint, so are the supports of the η_j. Further analysis of the run $\pi(\xi)$ also shows that $\forall i < j, target(\eta_i) \cap support(\eta_j) = \varnothing$.

If β has type $\sigma \multimap \sigma$, that means that in particular $\xi_0 = \xi_n$. This entails the following equality:

$$\bigcup_j support(\eta_j) = \bigcup_j target(\eta_j).$$

(The remaining places are those untouched by the execution.) Hence:

$$\begin{aligned} target(\eta_1) \subseteq \bigcup_j target(\eta_j) &= \bigcup_j support(\eta_j) \\ &= support(\eta_1) \cup \bigcup_{1<j} support(\eta_j) \\ support(\eta_1) \subseteq \bigcup_j support(\eta_j) &= \bigcup_j target(\eta_j) \\ &= target(\eta_1) \cup \bigcup_{1<j} target(\eta_j) \end{aligned}$$

We know that $\forall 1 < j$, $target(\eta_1) \cap support(\eta_j) = \varnothing$, hence $target(\eta_1) \subseteq support(\eta_1)$. To get the other direction, one has to notice that the run $\chi_2 \cdots \chi_p$ may be executed starting from ξ_0 (rather than ξ'_1). This is again because of the possibility to exchange "independent" transitions. Safety of the automaton ensures that

$$support(\eta_1) \cap \bigcup_{1<j} target(\eta_j) = \varnothing.$$

Thus we obtain $support(\eta_1) \subseteq target(\eta_1)$, finally proving $support(\eta_1) = target(\eta_1)$. This in turn ensures that $\xi'_1 = \xi_0$, and enable us to repeat the proof for $\eta_2, ..., \eta_n$.

The last thing to do is to "type check" these elementary boxes. Let's detail the computation of the type β outputs when given a tree σ. Each connected component C_j is a bow tie of centre o_j, with inputs S_j. For each of those, there must be a sub-tree T_j in σ with leaves S_j (otherwise there would be no way of reducing $\sigma \odot \beta$ to a tree). We then remove that sub-tree, and replace it by the top-down tree rooted in o_j. Because all S_j are pairwise disjoint, the trees T_j are also pairwise disjoint (otherwise they would share some leaves). With this description of the type-checking, one realises that 1) each of the η_j contribute to the final type on disjoint subtrees 2) and each of them leaves the rest of the tree alone. Because $\forall j, S_j = support(\eta_j) = target(\eta_j)$ we can conclude that η_j replaces the tree with leaves S_j with the tree with leaves S_j, and thus preserves the type $\sigma \multimap \sigma$.

□